03

Generando un jwt

Transcripción

[00:00] Bien, ahora ya tenemos la libería instalada en nuestro proyecto, vamos a comenzar creando un servicio para generar nuestros tokens en JWT y aprender, pues como implementarlos en nuestro código.

[00:13] Para eso como ustedes ya saben, voy a crear una clase llamada TokenService. No la voy a agregar aun al Git. Bueno, aquí es service para que Spring lo escanee como un servicio y aquí lo que yo voy a hacer es crear un método llamado public String generarToken() Y por ahora sin parámetros. Y aquí yo debo retornar alguna cosa.

[00:40] No voy a hacer nada. ¿Por qué? Porque yo no sé cómo usar esta librería de Auth0. Auth0 es el proveedor, pero yo no sé cómo usarla aún. ¿Qué puedo hacer para eso? Aquí en la documentación de la librería, en el repositorio de GitHub como lo pueden ver aquí, tenemos la parte instalación y cómo lo vimos en el video anterior, tenemos la parte de cómo crear un token JWT y nos da una pieza de código que la podemos usar como ejemplo, como inspiración.

[01:09] Bueno, lo que yo voy a hacer es copiar este código y voy a entrar aquí. Entonces lo pego aquí, voy a importar, me está pidiendo por ejemplo la clase Algorithm de Auth0, ya lo estoy importando.

[01:26] Vemos que el algoritmo que está usando esto es un algoritmo RSA256, que lo que usa es una llave pública y una llave privada para hacer el proceso de firma del token. Entonces yo no quiero usar mi llave pública ni privada porque no las he generado. Para los que no son muy familiares con esto, una llave

pública, una llave privada son digamos dos strings criptografiados con el cual yo comparo la validez de un token.

[01:57] Por ejemplo todo lo que sea firmado con la llave pública puede ser decodificado con la llave privada y viceversa. Entonces es un proceso de generación de llaves, no quiero entrar en detalles por ahora en esto, no lo voy a usar.

[02:10] Como no voy usar esto, yo voy a cambiar el tipo de algoritmo por un algoritmo poco más sencillo que es un HMAC256. Ese algoritmo lo que me pide en lugar de dos llaves de un key pair, dos llaves de seguridad me pide un string a arreglo de bytes. Yo le voy a dar string que es lo más fácil, que es el secret.

[02:34] El secret viene a ser como digamos la contraseña para decodificar este token. No para decodificar el token en sí sino para validar la firma. Para validar que este toklen, por ejemplo es el secret de voll.med, es del 1 al 6, entonces ese token puede ser validado si el receptor conoce esta firma. Ya lo vamos a ver más adelante.

[02:57] Vamos a importar esta libertad de JWT de Auth0, porque había dos, una es de Spring, una es de Auth0, importamos la de Auth0. Importamos este exception y para que no quede el catch vacío, lo cual es un anti patrón, vamos a darle un throw new RuntimeException() solamente para no dejarlo vacío. Aquí tú puedes poner un logger, lo que creas conveniente.

[03:24] Para nuestro caso solamente un catch vacío. No está compilando porque tenemos que retornar algo, nos pide un string. Tenemos que retornar un string que sería este de aquí. Yo voy a borrar esto y aquí le voy a dar un return y con esto ya tengo mi código listo.

[03:42] Una observación aquí, es de muy mala práctica tener secrets, y por secrets me refiero a llaves criptográficas, a contraseñas, API keys o lo que sea, en el código hardcodeadas, es muy mala práctica porque esto eso nunca debe ser compartido en ninguna fuente de código.

[04:05] Existen diferentes formas de obtener los secrets. Vamos a ver más adelante, pero por ahora por fines didácticos lo vamos a dejar así. Vamos a explorar un poco aquí sobre los parámetros del JWT. Aquí me dice JWT.created. Lo va a cargar con Issuer. Issuer es el que emite el JWT, en este caso por defecto llegó a Auth0, pero como este token no lo va a emitir AUth0 sino lo va a emitir "voll med", vamos a decirle "voll med".

[04:35] Y también vamos a decirle que va dirigido, entonces vamos a ponerle .withSubject()y por ahora vamos a hardcodearlo a "diego.rojas" Esto va a ir hardcodeado, la idea es hacerlo dinámico más adelante.

[04:51] Guardamos y vamos a probar si de verdad está funcionando nuestro método. Venimos aquí, mandamos el request, vemos que funciona. No devuelve nada, porque aún no llamo al token service desde mi controller. Entonces, lo siguiente que tenemos que hacer es entrar aquí a AutenticacionController.

[05:40] Voy a darle aquí un private TokenService. Vamos a importar el TokenService que hemos creado nosotros, porque hay dos. Uno es de Spring, otro es el que acabamos de crear. Importamos el nuestro y aquí lo que vamos a decirle es token = tokenService, esto no de aquí sino la instancia, tokenService.generateToken();

[06:09] Y aquí vemos que nos está dando un error porque token ya fue declarado. Lo que yo voy hacer para darle un mejor entendimiento a mis variables, voy a renombrar. Esta, voy a refactorizar aquí a authenticationToken, este de aquí va a ser mi JWTtoken. Mismo proceso, renombramos JWTtoken.

[006:38] Y entonces con el JWTtoken, lo que yo voy a retornar aquí en mi body el token que acabo de crear. Ya no necesito el build. Guardamos. Limpiamos esto de aquí, vamos a ver si va a funcionar, si nos va a retornar el token en efecto, esperamos aquí, vemos que mi servidor ya reinició, venimos aquí y ejecutamos.

[07:04] Vemos que me está dando un internal server error. Esto es quizá, porque aún no recarga, vamos a ver qué es lo que está pasando aquí. Venimos acá y me va a decir que no puede porque this tokenService is null. ¿Esto es por qué? Aquí me faltó un @Autowired. Ya tenemos el @Autowired.

[07:23] Guardamos, limpiamos nuestra terminal, nuestro tokenService está marcado como service, entonces no va haber ningún problema esta vez. Vemos que ya cargó, lanzamos y vean aquí lo que recibimos ahora. Ahora estamos recibiendo nuestro token en formato de JWT y vamos a validarlo. ¿Cómo lo validamos?

[07:49] Entramos aquí al sitio web de JSON web token, vimos que tienen los datos por defecto y el token que tú ves aquí lo que vas a hacer es borrarlo, y vas a pegar el que te acaba de generar tu propio API.

[08:01] Y en efecto miren aquí, tienes el tipo JWT con el algoritmo que hemos elegido el 256, y la información que hemos puesto aquí, el subject, o sea el usuario a quien va dirigido “diego.rojas” y el issue es “voll med”.

[08:21] En este caso de la firma me dice que es invadida porque esto está vacío, pero por ejemplo si yo pongo 123456, vamos a ver que la firma entonces ya pasa a ser verificada. Ese es el propósito del secret, verificar la firma. Entonces ahora ya sabes que el token realmente fue emitido por voll med.

[08:46] Entonces esto es alto nivel cómo funciona la librería de Auth0 JWT, pero aquí hay algunas cosas que tenemos que mejorar. La primera es que esto tiene que ser dinámico porque no todos los usuarios son Diego Rojas y esto no puede estar en el código porque es un secret y ya vimos que es muy mala práctica tenerlo aquí dentro del código. En el siguiente video vamos a ver cómo aplicar mejoras a este código. Nos vernos.